.NET Full Stack Development Program

—

Day 30 Agile & Scrum & Git & TFS & Swagger

Outline

- Project Management
  - SDLC & Agile & Scrum
- Version Control
  - Git
  - TFS

What is SDLC?

SDLC stands for Software Development Life Cycle, also known as Application Development Life Cycle.

It’s a systematic process for planning, creating, testing, and deploying

Why SDLC?

Produce high-quality software that meets or exceed customer expectations

Reach completion within time and cost estimates

With appropriate SDLC model, developers are able to:

- Give correct estimate on time & cost
- Organize efficient communication both inside team or between teams
- Increase work transparency, reduce chaos
- Help all the members better evaluate their performance

SDLC


Requirement Analysis
Design
Implementation
Testing
Evolution
SDLC
Software / System Development Life Cycle - SDLC

SDLC - Basic Steps

- **Requirement Analysis**
    - What is the purpose of this application?
    - What are the functionalities it should have?
    - Is it possible to reduce the workload by reuse existing application/service?
- **Design**
    - Macro-level
        - What programming language should we use?
        - Is our application stand alone? Will it interact with other applications?
        - If not stand alone, is there any standard we must follow?
        - What type of framework should we choose?
    - Micro-level
        - How many modules should we have? What are their responsibility?
        - How should we separate the codes and functions into different logic layers?
        - What coding/testing standard should we follow?
        - What are the design patterns the entire team must follow?

SDLC - Basic Steps

- **Implementation**
    - Manage dependencies
    - Writing code
    - Implement unit test/integration test
- **Testing**
    - Perform local testing
    - Deploying to testing environment
    - Find/fix bugs
    - Deploy to production environment
- **Evolution**
    - Review newly launched features; does it work good?
    - Is there anything that can be improved?

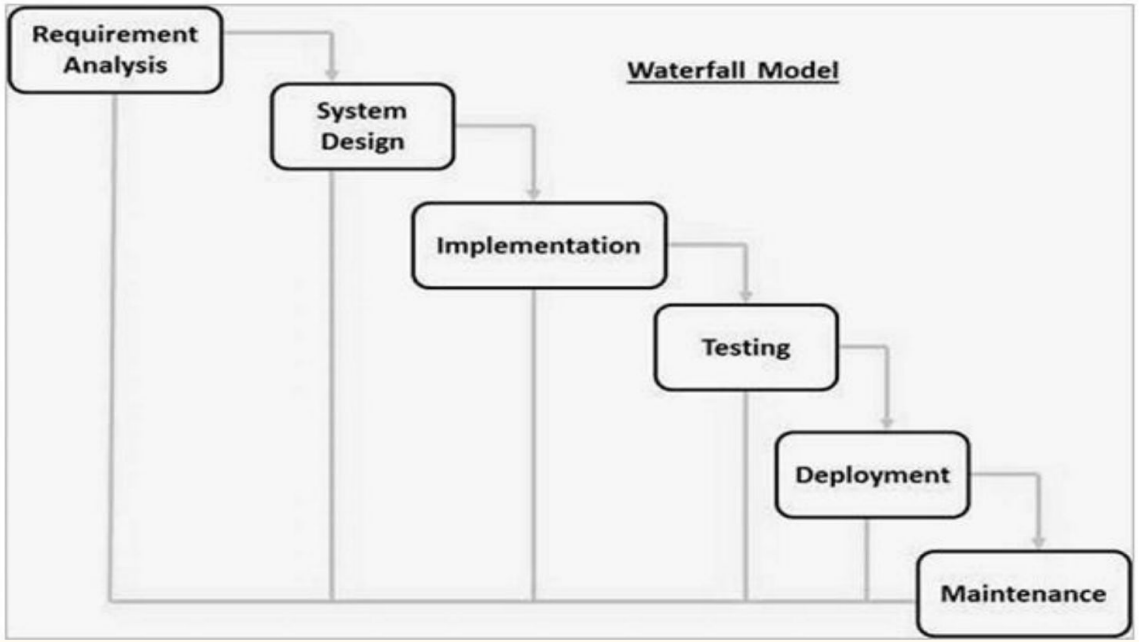
SDLC Models

- Agile
- Waterfall
- Iterative
- Spiral
- V-Model
- Big-Bang

Waterfall Model

- It is the earliest SDLC approach.
- Each phase must be completed entirely before starting the next phase
- When to use?
    - Requirements are well documented
    - Product definition is stable, technology is not dynamic
- Pros
    - Clear definition, simple and easy to understand
- Cons
    - Vulnerable to risk and uncertainty, hard to measure progress

Waterfall Model



Agile Model

Agile is a project management philosophy that utilizes a core set of values or principles

A SDLC model focuses on customer satisfaction with rapid delivery as its core idea.

• Breaks product into small incremental pieces/builds, divide and conquer in different iterations

• Each iteration will last for 2 to 4 weeks, team will focus on pre-scheduled task and deliver the result at the end of the time period

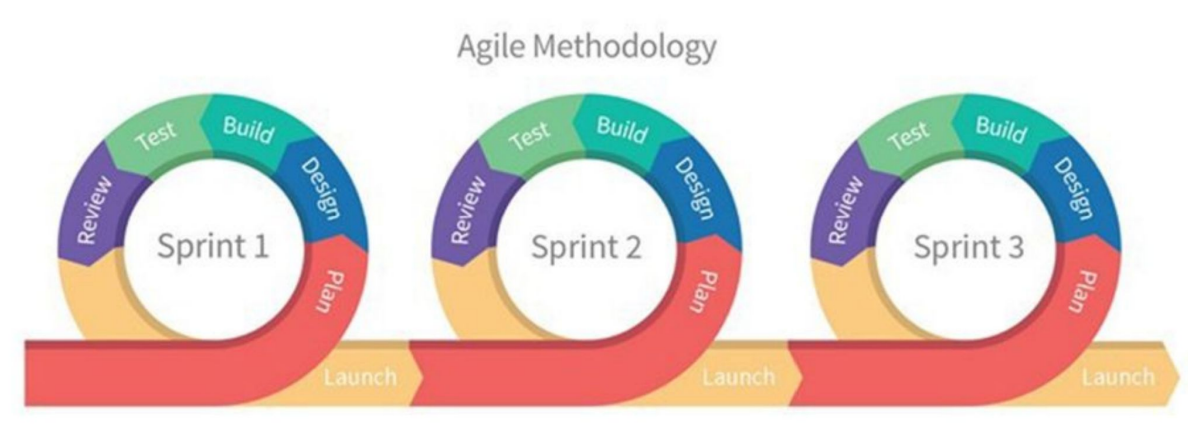
Agile Model

Pros

- Rapid development life cycle, require less resource
- Fast delivery, easy to manage
- Able to give accurate estimate on timeline

- Cons
  - No suitable for complex features
  - More efforts needed in maintenance
  - Leadership is required
  - Hard to transfer project knowledge as documentation is lacking

Agile Model


Agile Methodology
Review
Test
Build
Design
Plan
Sprint 1
Launch
Review
Test
Build
Design
Plan
Sprint 2
Launch
Review
Test
Build
Design
Plan
Sprint 3
Launch

Scrum

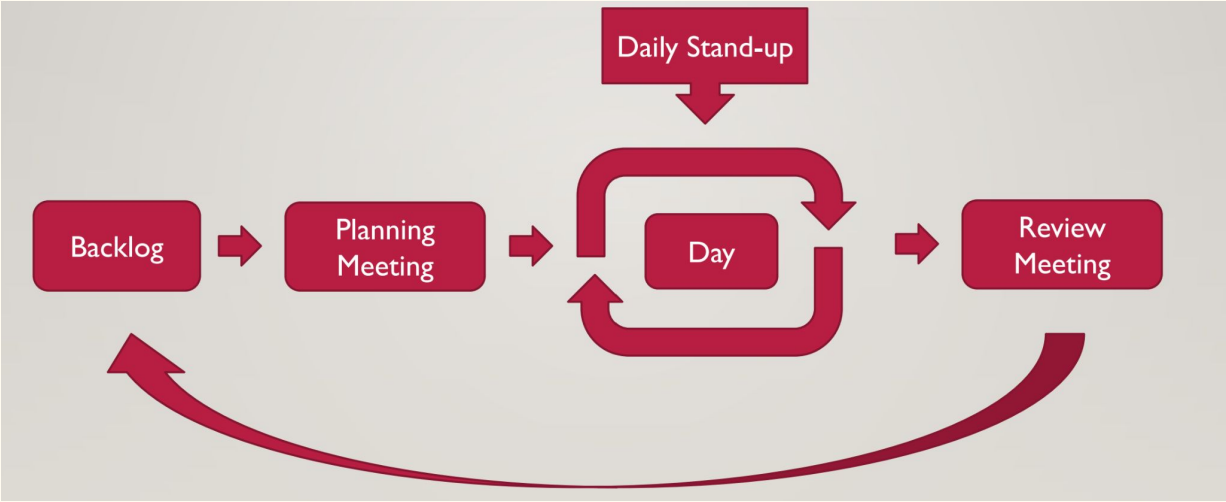
Scrum is a specific Agile methodology that is used to facilitate a project.

SDLC is principles, not actual implementation. Therefore, an actual practice or framework is needed to implement this idea — it is Scrum

A Scrum framework used by Agile principle, popular among enterprise level development team

- **Backlog**: place where we store all the information about our application/project, including tasks that need to be accomplished.
- **Sprint**: repeated fixed time-box. Development life cycle is composed of Sprints. Each sprint tasks will be assigned to team members.
- **Meeting**: in each sprint, the team will hold multiple meetings to plan for the tasks, solve issues, and improve overall performance

Scrum Workflow



Scrum Terminology

Important concepts in Scrum:

- Three Roles
- Three Work Pieces
- Five Events

Three Roles

Normally, an application will be assigned to a team. This team is in charge of design, maintain and make improvement.

To apply Scrum framework, a team must have the following part:

- **Product Owner**: owner of the product (application). Act as the bridge between tech team and business team. Help tech team understand the business requirement and help business team understand the tech efforts needed.
- **Scrum Master**: also known as Agile lead. Responsible for organizing meetings, checking team status and making sure everything is on schedule. Work closely with PO.
- **Develop team**: regular tech team. Usually consists of BE, FE and QE developers.

Three Work Pieces

Product Backlog: contains all the tasks of this application

Sprint Backlog: subset of Product backlog. Each Sprint, PO will pick tasks from product backlog and move into sprint backlog. So the team can narrow down the range and focus on certain tasks.

Increment: list possible improvements for existing features

Five Events

- **Sprint**: Sprint itself is an event, it composed of the following part
- **Sprint Planning Meeting**: Planning for the upcoming Sprint. Including narrow down the task range and assign the tasks to team members.
- **Daily Stand Up**: Daily report or your personal progress.
- **Sprint Review Meeting**: Review the completeness of the tasks in this sprint.
- **Sprint Retrospective Meeting**: Look back at the tasks, self evaluate and make improvement.

BackLog Management

- To manage backlog means we need to store our application information in certain format: word doc, Google doc, pdf, etc.
- **Software development tool – Jira, TFS, or Azure Boards**
- **Ticket** – Each task is called a ticket
    - Task type
    - Description
    - Comments

Development Environment

Software development usually have many different environment:

- **Local/Dev**: application deployed to local machine, laptop/desktop, for local testing
- **Staging**: deployed to remote cloud service, testing in real environment; for internally use only.
- **Production**: live version, customers will use this endpoint. Everything should be tested.
- **Pre-Production/Testing**: also for internally use, but environment is pretty much similar to production.

Development Environment

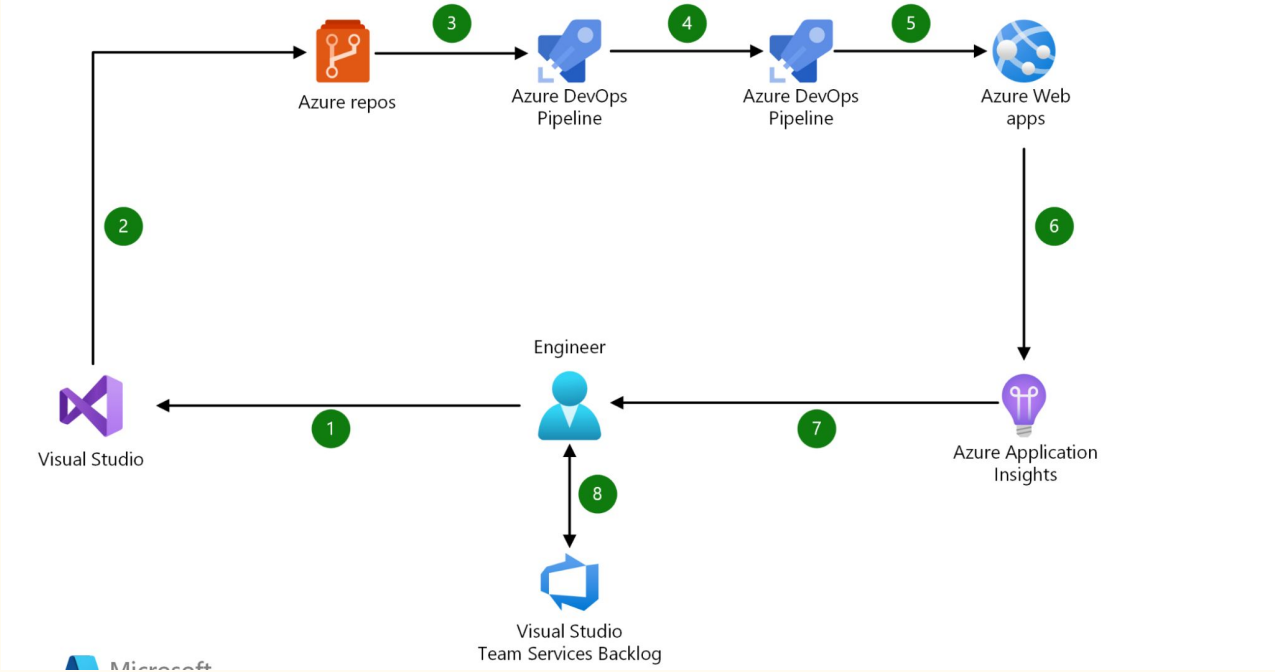
- Software development usually have many different environment:
    - Local: localhost:8080
    - Staging: www.testing.ebay.com
    - Production: www.ebay.com
    - Pre-Production: www.preprod.ebay.com

CI/CD Pipeline

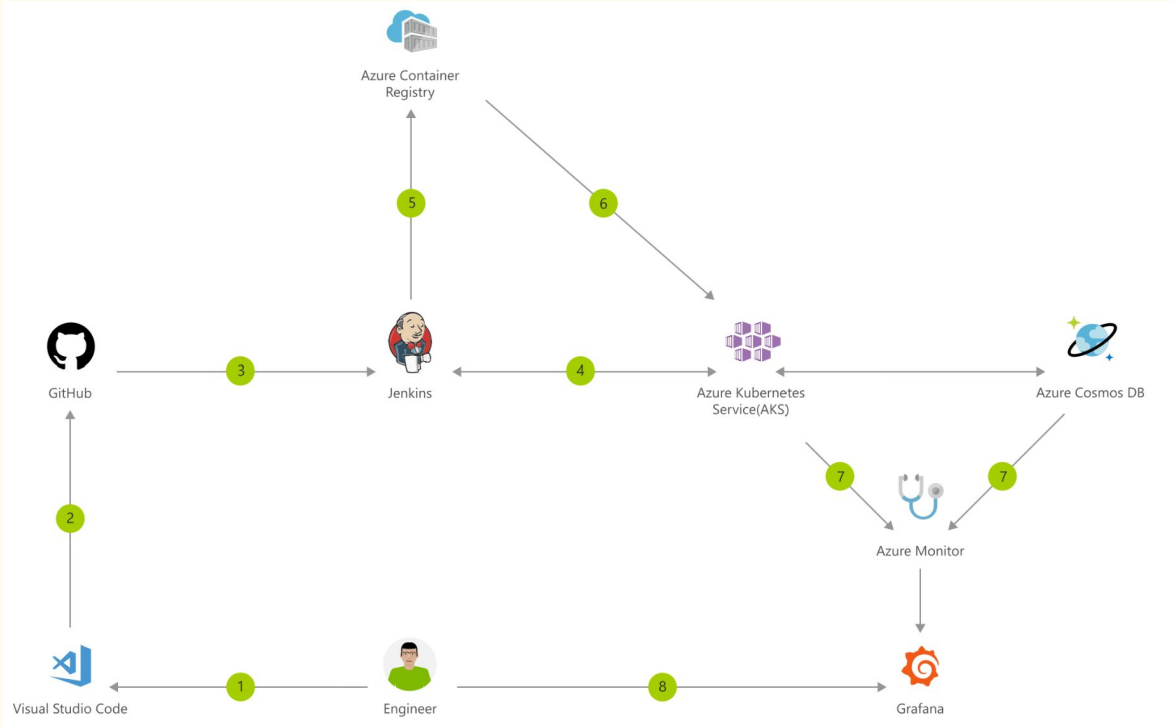
Continuous Integration and Continuous Delivery (CI/CD) is a method to frequently deliver apps to customers by introducing automation into the stages of app development.

Eg. **Azure DevOps Pipeline, Jenkins**, and etc

Example with Azure DevOps Pipeline



Example with Jenkins



What is Version Control

- Also known as **source control**
- What does version control do?
    - Helps us keep track of file changes
    - Provides metadata for each file change (date, description)
    - Allows us to switch between different file versions
    - Provides an easy way to rollback file changes

Tools used for version control: **GIT, Azure Repos** (Azure DevOps/TFS), and etc

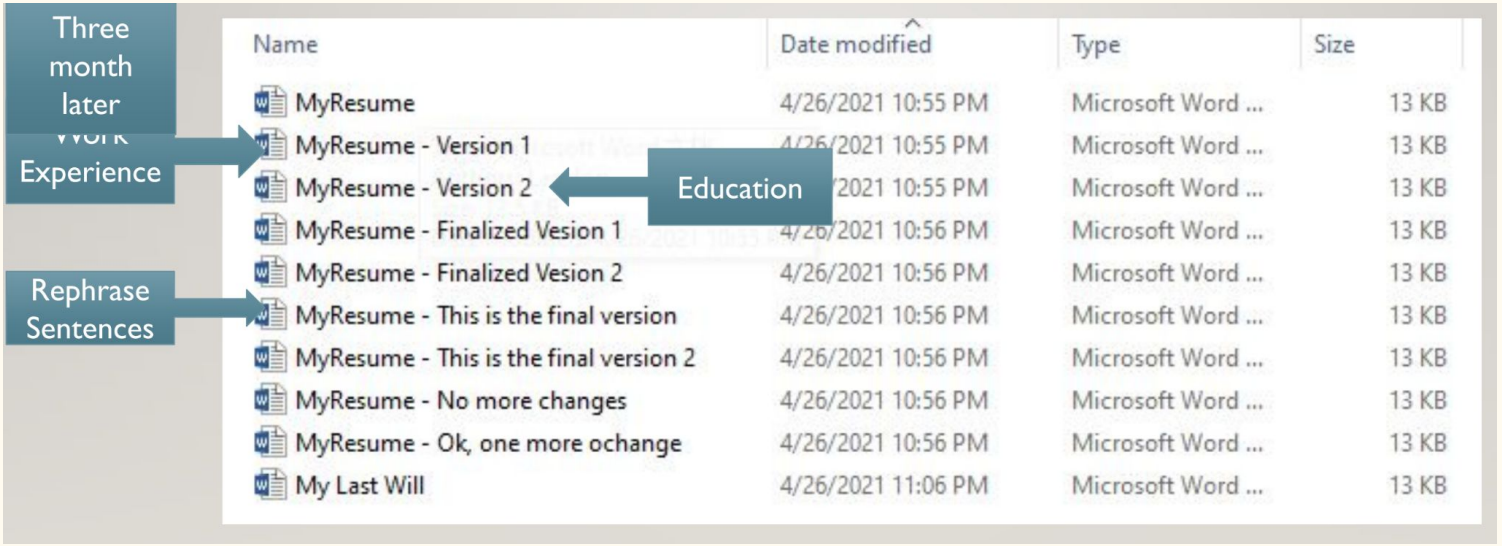
What is GIT

• “Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.”

• Launched in 2005 by Linus Torvalds to allow other developers to contribute to the Linux kernel.

• Git is a 3rd-generation (distributed) version control system, while most of Git’s predecessors are 2nd-generation (centralized) version control systems.

Version Control



Version Control

Version	File Name	User	Description	Date	FileLocation
1	UserController	Jason	Add getUserById() method	3/26/2021	C:\project\java\controller\UserController
2	UserController	Landon	Add getAllUsers() method	3/27/2021	C:\project\java\controller\UserController
3	UserController	Jason	Add getUserByName() method	3/28/2021	C:\project\java\controller\UserController
4	UserController	Landon	Add createNewUser() method	3/29/2021	C:\project\java\controller\UserController

Git Repository

The basic structure of a version control is a **Repository**

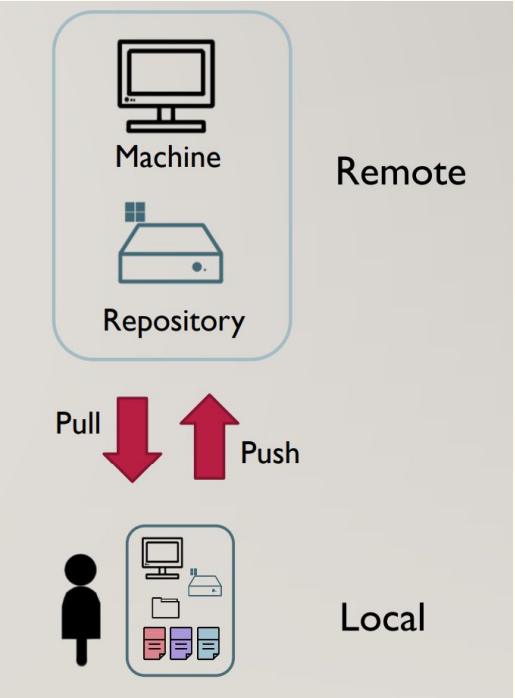
- Repository
    - Works like a database
    - Stores all the directories and its files
    - Stores the status of these directories
    - Stores the history record of all changes users made to the directories

Git Remote Repository

Remote repository work as a file exchange center

- We can push our changes from local repo to remote repo
- We can pull the changes from remote repo to our local repo

GitHub is an open source, public remote service.



Git Command - Initialize Local Repo

- Each application in Git has its own version storage called Repository
- **Repository** stores all the versions of files, allow us to switch between different versions
- There are two ways to create git repository:
    - **Clone** an existing one from remote repo to local

        Eg. **git clone {repo address}**
    - **Create** a brand new repo in local
- Initialize new local repository
    - Command: **git init**
        - Give git permission to manage our project
        - Create a ".git" folder with all the configuration files

Git Command – Inspect Repo

- Check current repo status: **git status**

```
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

  - On branch master – show which branch we are currently on
  - No commits yet – we did not commit any file to our version control system
  - Nothing to commit – we do not have any file that need to be committed

Git Command – Inspect Repo

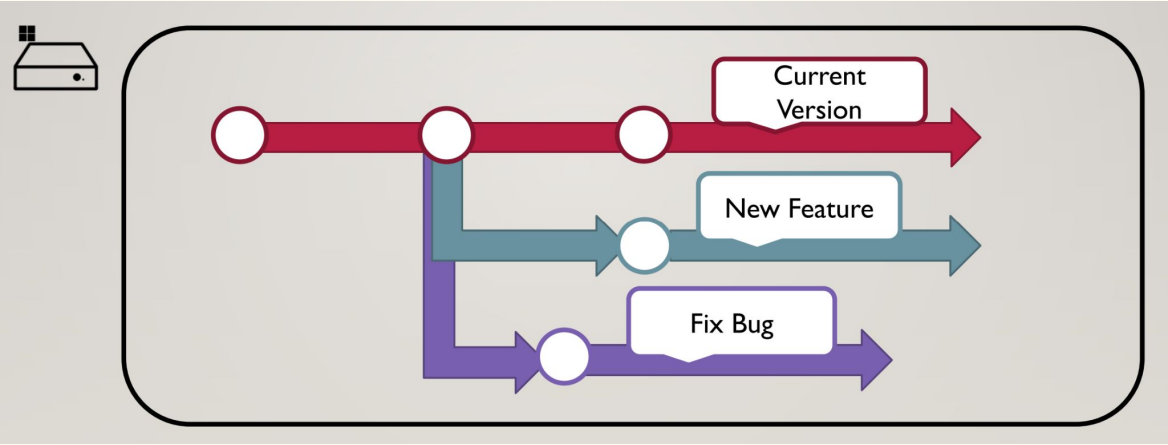
- Once we add new files or modify/delete existing files, we will have untracked files
  - Git will tell us we have untracked files/changes that need to be taken care of

```
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Example.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Git Command - Create Your Own Branch

- What is branch?
    - A feature available in most modern version control system
    - An independent line of development.
- Why do we need branch?
    - Isolate the development of new feature from existing code base
    - Prevent unstable code to be merged into main code base

Git Branch Example



Git Command – Create and Switch Branch

- Inspect Branch
    - **git branch -v**
- Create Branch
    - **git branch {name}**
- Switch between Branches
    - **git checkout {name}**

Git Command – Add or Remove Files

• To save our modification to the code repository, we first need to add the changes to staging area

- **git add {filename}**
- **git add .**

• If we found something is wrong with current modification and we do not want to save the file into version history, we can also remove the file from staging area

- **git rm --cached {filename}**
- **git rm -r --cached .**

Git Command – Commit to Repository

- Command
  - Specific file: **git commit {filename} –m “{logging message}”**
  - All files: **git commit –m “{logging message}”**
- Git will submit file/files from staging area to the repository and create a version history
- Logging is a short description of the current commit, it helps document the changes and helped other user understand the content.

Git Commands - Interact with Remote Service

- Fetch remote changes
    - **git pull**
    - **git rebase**
- Submit local changes
    - **git push**

Merging Branches

- When development is finished and tested, we would like to merge our changes back into our main branch
- Merge Branches
    - **git merge {name}**
    - For example, if we want to merge a feature branch into the master branch, we will first switch to the master branch then execute git merge feature
    - Merging may cause conflicts (merge conflict)

Merging Branches (cont.)

In enterprise project, you need to create a **pull request (pr)** in order to merge your branch into main branch. On that pull request, you will be asked to tag a couple of code reviewers to review your code. Once your code is approved by reviewer, you can then merge your code based on the procedure

Merge Conflict

- Happens when two developers make changes on the same file, in the same location
- Git does not know which change it should take as the version to use
- How to resolve a merge conflict?
    - Open the files in editor and manually fix the conflict
    - Commit the change

Avoid Conflict

• Merge conflicts are most likely to happen when multiple developers work on the same file

• To avoid conflict:

  • **Divide & conquer** – Split task to each member will work on different parts

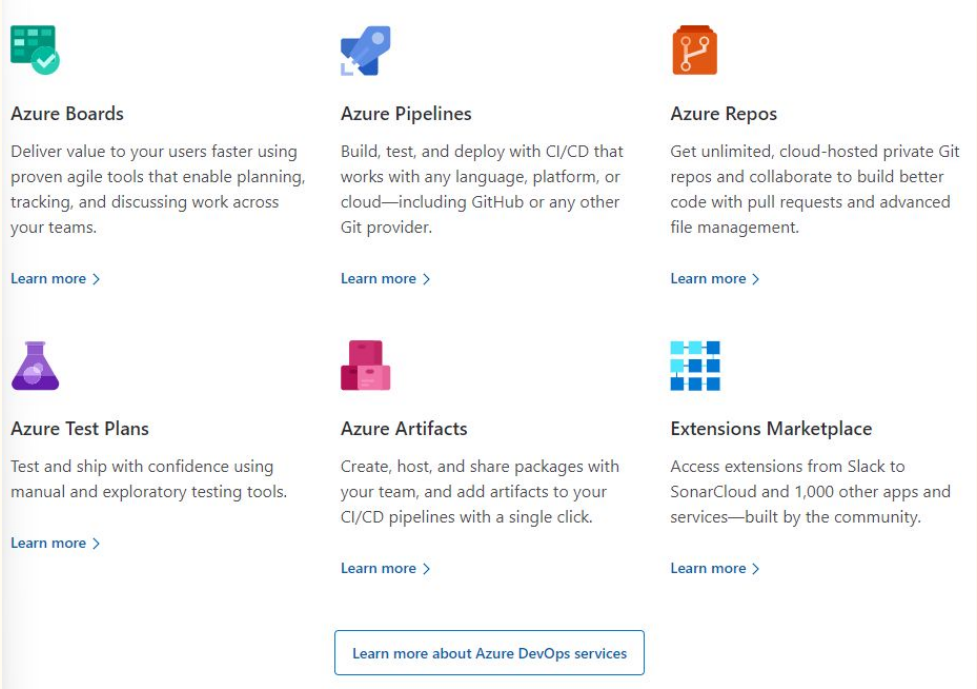
  • **Communicate** – When you need to make changes on other files, always notify your team member

What is TFS?

TFS stands for the Team Foundation Server. It’s an old version of Collaborative software development tools for the entire team

Previously known as Team Foundation Server (TFS), **Azure DevOps Server** is a set of collaborative software development tools, hosted on-premises. Azure DevOps Server integrates with your existing IDE or editor, enabling your cross-functional team to work effectively on projects of all sizes.

Azure DevOps Services



Questions?